Real Time Linux Workshop, Vienna University of Technology, Dec. 1999

DIAPM RTAI for Linux: WHYs, WHATs and HOWs

Paolo Mantegazza

Dipartimento di Ingegneria Aerospaziale
Politecnico di Milano

REAL TIME AT DIAPM

Data acquisition

Digital implementation of complex control systems (MIMO, neural, fuzzy, adaptive)

Test rigs control and monitoring (static, dynamic, fatigue)

Realtime simulation with hardware in the loop

high performances

GOOD QUALITY

LOWEST COST

possibly FREE

Under the constraint:

ALWAYS DO MUCH MORE WITH MUCH MORE LESS

NOTHING BUT TIME IS *FREE*

WE ALL KNOW THAT TIME IS MONEY

<u>BUT</u>

IF YOU DO NOT SLEEP

AND DO NOT GO ON VACATIONS

TIME BECOMES A *FREE*

(ALMOST) UNLIMITED RESOURCE

Real Time Experience at DIAPM

PC + <u>DOS</u>

- humble slave to boot and for basic services;
- easy in surrendering the hardware back and forth with simple user interfacing;
- <u>PC fully available in real mode</u>.

16 bits / real mode

Some exposure to structured approaches (**QNX, RTKernel, UCOS**) resulted in

PCDOS-DIAPM-RTOS

or

TERMINATE and STAY RESIDENT ***(TSR)*** technique

The mostly used

32 bits real/protected mode

possibly better

under DOS: more difficult to find free, or almost free, software;
under Windows: even more costlier and possible loss of performance;
with Ppro 16 bits ran slower than on Pentiums

first solution devised

Port of ***DIAPM-RTOS* *and TSR*** in
FULL 32 bits real time real/protected mode
using GNU-DOS

Search for "free" "structured" high performance

•**RTMach** and **Maruti** (too complex)

the most viable

• **DOS**+**RTEMS** (to be explained later)

• **RTHAL-RTAI under Linux** (recovering all what available with DOS/16 bits)

BUT

• Approaching Linux, around 2.0.25, for RTHAL-RTAI appeared too complex (interaction with the hardware too scattered, too many cli/sti,…)

•After deciding to go to try
DOS+RTEMS vs DIAPM-RTOS

NMT-RTL appeared

NMT-RTL

• *NMT-RTL patch confirmed that 2.0.xx was not mature for RTHAL-RTAI;*

• *Its simple scheduler, declared as primitive by NMT-RTL developers, was instead immediately recognized as what we needed, because it was very close to that of DIAPM-RTOS;*

• *So we could easily go to "**the old loved DOS way**" and easily port all what we had under DOS (DIAPM-RTOS almost unchanged, TSRs became LINUX modules);*

BUT ...

The first tests were a disaster!!!

(on a PPro it was not possible to get 486DX280 performances!)

The culprit was immediately spotted -> **ONE SHOT TIMING**
(about 10 us to program the timer!!!)

THAT'S WHY

DIAPM-RTL VARIANT

WAS IMMEDIATELY BORN

DIAPM-RTL VARIANT

Maintained only the NMT-RTL *kernel patch*

- The RTL scheduler base remained (very close to that of DIAPM-RTOS) but 90% of DIAPM-RTOS services were recovered (almost completely as they were under DOS) and added anew:
  - ***semaphores*** (rt_sem_init, rt_sem_signal, rt_sem_wait, rt_sem_wait_until, rt_sem_wait_if,…)
  - ***intertasks messages*** (rt_send, rt_receive, rt_send_timed, …)
  - ***intertasks messages "a la QNX"*** (rt_rpc, rt_return)
  - ***timing services*** (rt_sleep, rt_busy_sleep, *time conversion functions*, …)

- Fixed the <u>floating point support</u>
- Modified all what was related to the <u>real time timing</u>:
  - introduction of a <u>***periodic timing***</u> to enhance efficiency in control applications, when one can work with a basic period and integer multiples of it
  - recovered ***oneshot timing*** anew by using the CPU <u>***TSC***</u> (<u>*Time Stamp Clock*</u>) with far greater efficiency (only 2 I/O instructions to 8254 instead of 9 of RTL) (not usable with earlier than Pentium machines, and compatibles).

BIRTH OF DIAPM-RTAI

End of 1998

Feasibility study for adaptive secondary mirror real time reconstructor:

SOLUTIONS

CUSTOM DSP BOARDs

VS

DUAL SMP 400 MHz PC with **PCI CCD camera frame acquisition** and **custom (MICROGATE) 200 Mbs high speed fiber link** to decentralized controllers

a order of magnitude difference in cost !!!

- control frequency **1000 Hz** (max latency/jitter 50 us);
- ≅ **100 Mflops** (tight mul/add loop);
- send commands-receive states to/from about 170 massively decentralized active structural controllers using 170: AD-2181 DSPs with two16 bits ADC and 16 bits DAC, each one controlling two linear magnetic actuators at 40 KHz
- logging **5 MB/s** to disk;
- under moderate front end activity.

Beginning of 1999 ⇒ ***FROZEN SITUATION:***

- *2.2.xx available and simple to patch the* RTHAL-RTAI *way;*
- *both* DIAPM-RTL *variant and* NMT-RTL *did not support* SMP*;*
- *no sign from* NMT-RTL *of a prompt stable upgrading of* RTL *to 2.2.xx and* SMP.

Since the skeleton of RTHAL-RTAI *was already tested and verified, the decision was taken to* ***<u>GO AHEAD</u>*** *and verify that the mirror control problem could be solved*.

Middle of February ⇒

RTHAL-RTAI *was reliably proving that the mirror control specs could be satisfied on a dual 350 PII and using* RTL-*like fifos*

Beginning of March ⇒

Posting of a ***public call*** *to the* RTL *newsgroup and to TORVALDS to join under a <u>common</u> <u>development</u> for* RTL-*2.2.xx* UP-SMP real time *based on the* RTHAL-RTAI *concept resulted in:*

- *discussion on a better organization of RT-Linux directory tree;*
- ***refusal*** *of* NMT-RTL *to join on a common* RTHAL-RTAI*;*
- *no answer from TORVALDS.*

Middle of March

A check at NMT-RTL *delayed development resulted in the decision to go on also in porting all what available under the* DIAPM-RTL *variant to the* RTHAL-RTAI *substrate ...*

... EVEN IF THIS COULD MEAN STAYING ALONE FOREVER, BUT WITH THE POSSIBILITY OF TAKING WHATEVER USEFUL WOULD HAVE COME FROM FUTURE RTL DEVELOPMENT

Beginning/Middle of April

- *All what available under* DIAPM-RTL *variant recovered both for* UP *and* SMP *under an* SMP *compiled kernel*
- ***First version** of all the stuff under the acronym* **RTAI** *released*

What is RTHAL?

The RTHAL *performs three primary functions:*

- *gather all the pointers to the required internal data and functions into a single structure, rthal, to allow an easy trapping of all the kernel functionalities that are important for real time applications, so that they can be dynamically switched by* RTAI *when hard realtime is needed;*

- *makes available the substitutes of the above grabbed functions and sets rthal pointers to point to them;*

- *substitutes the original function calls with calls to the rthal pointers in all the kernel functions using them.*

Linux is almost uneffected by RTHAL, except for a slight (and negigible) loss of performance due to calling cli and sti, and flags related functions, in place of their corresponding original Linux function calls and macros.

About 70 lines of code is all of what is changed/added in the kernel.

RTHAL Structure

```
struct rt_hal {
        struct desc_struct *idt_table;
         void (*disint)(void);
         void (*enint)(void);
         unsigned int (*getflags)(void);
         void (*setflags)(unsigned int flags);
         void (*mask_and_ack_8259A)(unsigned int irq);
         void (*unmask_8259A_irq)(unsigned int irq);
         void (*ack_APIC_irq)(void);
         void (*mask_IO_APIC_irq)(unsigned int irq);
         void (*unmask_IO_APIC_irq)(unsigned int irq);
        unsigned long *io_apic_irqs;
        void * irq_controller_lock;
        void *irq_desc;
        int *irq_vector;
        void *irq_2_pin;

        void *ret_from_irq;
};
```


What is RTAI?

It is a module in dormant state ready to overtake Linux

RTAI init_module *does a few important things:*

- *initializes all of its control variables and structures;*
- *makes a copy of the* idt_table *and of the Linux irq handlers entry addresses;*
- *initializes the interrupts chips management specific functions.*

rtai.h *contains basic defines and inlined functions which performs* <u>RTAI *services*</u>

timers services 8254 *timer and* APIC *timers*

when one does

rt_mount_rtai *hardware fully trapped !!!*

RTAI services

•*implementation of a **specific lock service** (Linux spinlocks are no more protected by disabling the interrupt flags as Linux hold just soft flags, while* RTAI *needs true disables)* →

unsigned long flags, spinlock_t lock, rt_spin_lock(&lock), rt_spin_unlock(&lock), rt_spin_lock_irq(&lock), rt_spin_unlock_irq(&lock), flags=rt_spin_lock_irqsave(&lock), rt_spin_lock_irqrestore(flags,&lock)

• *implemention of a **global lock service** (to obtain atomicity across* CPUs*)* →

unsigned long flags, rt_global_cli(), rt_global_sti(), rt_global_save_flags(), flags=rt_global_save_flags_and_cli(), rt_global_restore_flags(flags)

• *implementation of a special form of hard lock disable across* CPUs →

unsigned long flags, flags=hard_lock_all(), hard_unlock_all(flags)

RTAI mounting

• sets up the global hard lock handler;

• hard locks all CPUs;

• redirects rthal interrupts enable/disable and flags save/restore to its internal functions doing it all in software;

• recovers from rthal a few functions to manipulate 8259 PIC and IO_APIC mask/ack/unmask staff;

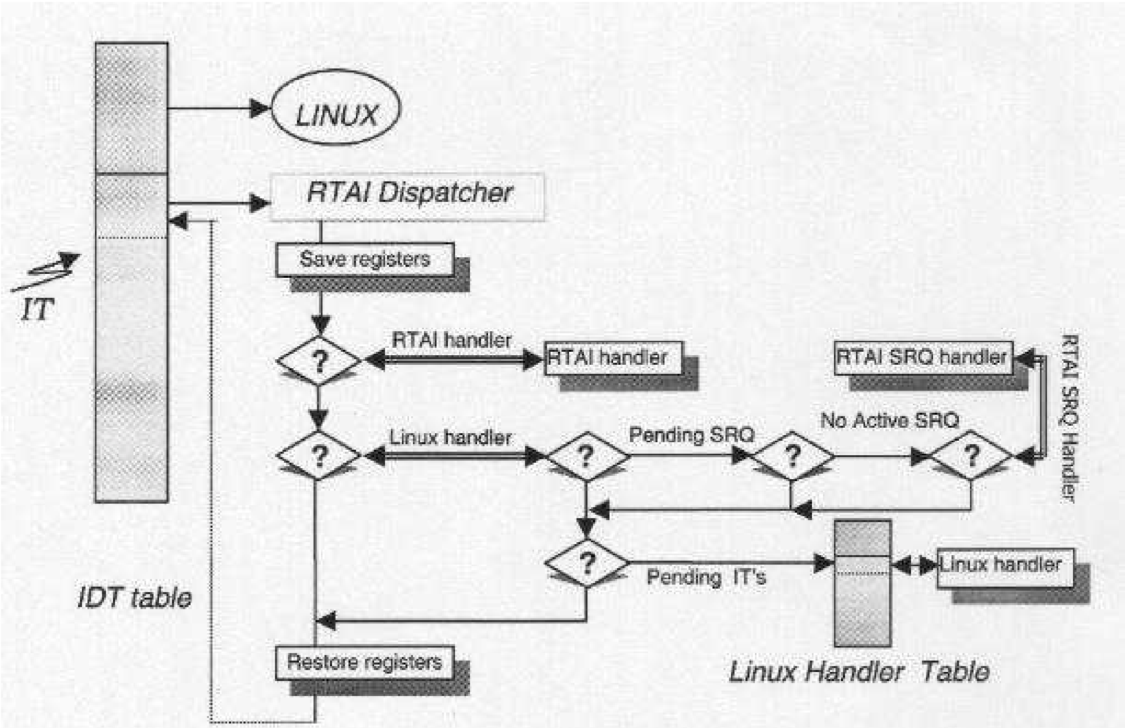
• redirect all hardware handler structures to its trapped equivalent;

• changes the handlers functions in idt_table to its dispatchers;

• releases the global hard lock.

Linux appears working as nothing happened
but it is <u>no more</u> the machine master

RTAI interrupt dispatcher



(from RTAI Internals Presentation, by Patrick Mourot, ALCATEL FRANCE)

TIMERS

8254

ONE CHIP PER BOX

3 COUNTERS

usable for UP and SMP

	LINUX	***RTAI***
counter 0	*used to pace periodically at* param.h HZ *macro (usually 100 hz) (related counter: linux jiffies)*	*used for oneshot / periodic mode*
counter 1	*not available (used for RAM refresh cycles?)*	
counter 2	*used for beeping frequency*	*used to emulate TSC on 486 boxes (beeping muted …)*

Linux time is kept by pending a Linux interrupt at due time

usable for SMP and MUP

Local APIC

ONE PER CPU
(unfortunately not available on UP)

LINUX: *for profiling and scheduling (?)*

RTAI: *uses just one or all either for homogeneous or heterogeneous periodic / oneshot*

Linux LOCAL APIC timers kept by broadcasting to the Linux Local APIC timer handlers at each jiffy

TIMERS & CONTROLLERS

Timers alone (in modules coupled to FIFOs and Shared Memory modules) are the basic approach to implement high performances controllers (recall the mirror starting point…), **the controller being the timer interrupt handler** (or handlers with MP). For this RTAI has available:

- rt_request_timer (handler, tick, choose_apic_8254)
- rt_request_apic_timers (handler, apic_timer_data)

The single APIC timer of rt_request_timer interrupt is installed on the CPU that executes the function, while the 8254 timer interrupt can be directed to any desired CPU with rt_assign_irq_to_cpu (*Linux uses Simmetric Delivery*).

At the moment only a single handler can be assigned in rt_request_apic_timers; adding multiple handlers is trivial.

RTAI MODULES

- SHARED MEMORY
- FIFOS and SEMAPHORES (with no reat time schedulers installed)
- REAL TIME SCHEDULERS (UP, SMP, MUP)
- POSIX API
- LXRT (inter-intra Linux-RTAI support module)

SHARED MEMORY & FIFOS

Timers are just RTAI interrupt handlers. **To communicate with LINUX process**, RTAI makes available:

SHARED MEMORY

It is a friendly user API, just malloc and free, based on T. Motylevsky 'mbuf-kvmem' substrate (derived by hacking Linux bttv.c). (RTAI shared memory API was developed to avoid the misteries and intrecacies that made mbuf usable only by wizards…). RTAI shared memory can be used inter-intra Linux processes/modules

FIFOs

In RTAI they are implemented as <u>mailboxes</u>, allowing only non-blocking operations from the module-side and blocking operations from Linux processes. They include <u>semaphores</u> that are useful for synchronization, e.g. coordinate shared memory operations.

FIFOs Services

used in RTAI modules

int **rtf_create_handler** (unsigned int fifo, int (*handler)(unsigned int fifo))
int **rtf_create** (unsigned int fifo, int size)
int **rtf_reset** (unsigned int fifo)
int **rtf_destroy** (unsigned int fifo)
int **rtf_resize** (unsigned int minor, int size)
int **rtf_put** (unsigned int fifo, void *buf, int count)
int **rtf_get** (unsigned int fifo, void *buf, int count)
int **rtf_sem_init** (unsigned int fifo, int value)
int **rtf_sem_post** (unsigned int fifo)
int **rtf_sem_trywait** (unsigned int fifo)
int **rtf_sem_destroy** (unsigned int fifo)
int **rt_printk** (const char *fmt, ...)
int **rt_print_to_screen** (const char *fmt, ...)

used in LINUX processes

int **open** (const char *pathname, int flags, mode_t mode)
ssize_t **write** (int fd, const void *buf, size_t count)
ssize_t **read** (int fd, void *buf, size_t count)
int **select** (int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
int **poll** (struct pollfd *ufds, unsigned int nfds, int timeout)
void **rtf_set_async_sig** (int fd, int signum)
int **rtf_reset** (int fd)
int **rtf_resize** (int fd, int size)
void **rtf_suspend_timed** (int fd, int ms_delay)
int **rtf_open_sized** (const char *dev, int perm, int size)
int **rtf_read_all_at_once** (int fd, void *buf, int count)
int **rtf_read_timed** (int fd, void *buf, int count, int ms_delay)
int **rtf_write_timed** (int fd, void *buf, int count, int ms_delay)
void **rtf_sem_init** (int fd, int value)
int **rtf_sem_wait** (int fd)
int **rtf_sem_trywait** (int fd)
int **rtf_sem_timed_wait** (int fd, int ms_delay)
void **rtf_sem_post** (int fd)
void **rtf_sem_destroy** (int fd)

SCHEDULERS

UP

- *only for uniprocessor (*DIAPM-RTL *variant scheduler unchanged)*
- 8254 *based*
- *oneshot/periodic timing*
- *486 compatible*

SMP

- *for multiprocessors*
- ***single** oneshot/periodic timing (either* 8254 *or* LOCAL APIC *based, the first one usable also with* UPs*)*
- *tasks can run symmetrically on any or a cluster of* CPUs, *or bound to a single* CPU

MUP

- *only for multiprocessors*
- *task must be bound to a single CPU*
- *a LOCAL APIC based timer per CPU (**heterogeneous timing possible**, i.e. mixing of different ticked periodic timers and/or oneshot)*

Scheduler Services

all the functions to be presented can be used with any scheduler (reasonable default actions taken for natively specific function; user can dinamically change the defaults afterward)

- ***task functions***
- ***timing functions***
- ***semaphore functions***
- ***intertask communication functions***

TASK FUNCTIONS

int **rt_task_init**(RT_TASK *task, void (*rt_thread)(int), int data,int stack_size,
int priority, int uses_fpu, void(*signal)(void))
int **rt_task_init_cpuid**(RT_TASK *task, void (*rt_thread)(int), int data, int stack_size,
int priority, int uses_fpu, void(*signal)(void)
unsigned int run_on_cpu)
void **rt_set_runnable_on_cpus** (RT_TASK *task, unsigned int cpu_mask)
void **rt_set_runnable_on_cpuid** (RT_TASK *task, unsigned int cpuid)
int **rt_task_delete** (RT_TASK *task)
int **rt_task_signal_handler** (RT_TASK *task, void (*handler)(void))
int **rt_task_use_fpu** (RT_TASK *task, int use_fpu_flag)
void **rt_linux_use_fpu** (int use_fpu_flag)
void **rt_preempt_always** (int yes_no)
void **rt_preempt_always_cpuid** (int yes_no, unsigned int cpuid)
void **rt_task_yield** (void)
int **rt_task_suspend** (RT_TASK *task)
int **rt_task_resume** (RT_TASK *task)
RT_TASK ***rt_whoami** (void)

TIMING FUNCTIONS (I)

int **rt_get_timer_cpu** (void);
void **rt_set_periodic_mode** (void);
void **rt_set_oneshot_mode** (void);
RTIME **start_rt_timer** (int period);
RTIME **start_rt_timer_ns** (int period)
RTIME **start_rt_apic_timers** (struct apic_timer_setup_data *setup_mode, unsigned
int rcvr_jiffies_cpuid);
RTIME **stop_rt_timer** (void);
RTIME **count2nano** (RTIME timercounts);
RTIME **nano2count** (RTIME nanosecs);
RTIME **count2nano_cpuid** (RTIME timercounts, unsigned int cpuid);
RTIME **nano2count_cpuid** (RTIME nanosecs, unsigned int cpuid);
RTIME **rt_get_time** (void);
RTIME **rt_get_time_cpuid** (unsigned int cpuid);
RTIME **rt_get_time_ns** (void);
RTIME **rt_get_time_ns_cpuid** (unsigned int cpuid);
RTIME **rt_get_cpu_time_ns** (void);

TIMING FUNCTIONS (II)

int **rt_task_make_periodic_relative_ns** (RT_TASK *task, RTIME start_delay, RTIME period);
int **rt_task_make_periodic** (RT_TASK *task, RTIME start_time, RTIME period);
void **rt_task_wait_period** (void);
RTIME **next_period** (void);
void **rt_busy_sleep** (int nanosecs);
void **rt_sleep** (RTIME delay);
void **rt_sleep_until** (RTIME time);

SEMAPHORE FUNCTIONS

void **rt_sem_init** (SEM *sem, int value);
int **rt_sem_delete** (SEM *sem);
int **rt_sem_signal** (SEM *sem);
int **rt_sem_wait** (SEM *sem);
int **rt_sem_wait_if** (SEM *sem);
int **rt_sem_wait_until** (SEM *sem, RTIME time);
int **rt_sem_wait_timed** (SEM *sem, RTIME delay);

INTERTASK COMM. FUNCTIONS

RT_TASK ***rt_send*** (RT_TASK *task, unsigned int msg);
RT_TASK ***rt_send_if*** (RT_TASK *task, unsigned int msg);
RT_TASK ***rt_send_until*** (RT_TASK *task, unsigned int msg, RTIME time);
RT_TASK ***rt_send_timed*** (RT_TASK *task, unsigned int msg, RTIME delay);
RT_TASK ***rt_receive*** (RT_TASK *task, unsigned int *msg);
RT_TASK ***rt_receive_if*** (RT_TASK *task, unsigned int *msg);
RT_TASK ***rt_receive_until*** (RT_TASK *task, unsigned int *msg, RTIME time);
RT_TASK ***rt_receive_timed*** (RT_TASK *task, unsigned int *msg, RTIME delay);
RT_TASK ***rt_rpc*** (RT_TASK *task, unsigned int to_do, unsigned int *result);
RT_TASK ***rt_rpc_if*** (RT_TASK *task, unsigned int to_do, unsigned int *result);
RT_TASK ***rt_rpc_until*** (RT_TASK *task, unsigned int to_do, unsigned int *result, RTIME time);
RT_TASK ***rt_rpc_timed*** (RT_TASK *task, unsigned int to_do, unsigned int *result, RTIME delay);
int **rt_isrpc** (RT_TASK *task);
RT_TASK ***rt_return*** (RT_TASK *task, unsigned int result);

POSIX Services

- *thread functions*
- *mutex functions*
- *condvar functions*
- *message queue functions*

THREAD FUNCTIONS

int **pthread_create** (pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)
(void *), void *arg)
void **pthread_exit** (void *retval)
pthread_t **pthread_self** (void)
int **pthread_attr_init** (pthread_attr_t *attr)
int **pthread_attr_destroy** (pthread_attr_t *attr)
int **pthread_attr_setdetachstate** (pthread_attr_t *attr, int detachstate)
int **pthread_attr_getdetachstate** (const pthread_attr_t *attr, int *detachstate)
int **pthread_attr_setschedparam** (pthread_attr_t *attr, const struct sched_param
*param)
int **pthread_attr_getschedparam** (const pthread_attr_t *attr, struct sched_param
*param)
int **pthread_attr_setschedpolicy** (pthread_attr_t *attr, int policy)
int **pthread_attr_getschedpolicy** (const pthread_attr_t *attr, int *policy)
int **pthread_attr_setinheritsched** (pthread_attr_t *attr, int inherit)
int **pthread_attr_getinheritsched** (const pthread_attr_t *attr, int *inherit)
int **pthread_attr_setscope** (pthread_attr_t *attr, int scope)
int **pthread_attr_getscope** (const pthread_attr_t *attr, int *scope)
int **sched_yield** (void)

MUTEX FUNCTIONS

int **pthread_mutex_init** (pthread_mutex_t *mutex, const pthread_mutexattr_t
*mutex_attr)
int **pthread_mutex_destroy** (pthread_mutex_t *mutex)
int **pthread_mutexattr_init** (pthread_mutexattr_t *attr)
int **pthread_mutexattr_destroy** (pthread_mutexattr_t *attr)
int **pthread_mutexattr_setkind_np** (pthread_mutexattr_t *attr, int kind)
int **pthread_mutexattr_getkind_np** (const pthread_mutexattr_t *attr, int *kind)
int **pthread_setschedparam** (pthread_t thread, int policy, const struct sched_param
*param)
int **pthread_getschedparam** (pthread_t thread, int *policy, struct sched_param
*param)
int **pthread_mutex_trylock** (pthread_mutex_t *mutex)
int **pthread_mutex_lock** (pthread_mutex_t *mutex)
int **pthread_mutex_unlock** (pthread_mutex_t *mutex)

CONDVAR FUNCTIONS

int **pthread_cond_init** (pthread_cond_t *cond, const pthread_condattr_t *cond_attr)
int **pthread_cond_destroy** (pthread_cond_t *cond)
int **pthread_condattr_init** (pthread_condattr_t *attr)
int **pthread_condattr_destroy** (pthread_condattr_t *attr)
int **pthread_cond_wait** (pthread_cond_t *cond, pthread_mutex_t *mutex)
int **pthread_cond_timedwait** (pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime)
int **pthread_cond_signal** (pthread_cond_t *cond)
int **pthread_cond_broadcast** (pthread_cond_t *cond)

MESSAGE QUEUE FUNCTIONS

mqd_t **mq_open** (char *mq_name, int oflags, mode_t permissions, struct mq_attr *mq_attr)
size_t **mq_receive** (mqd_t mq, char *msg_buffer, size_t buflen, unsigned int *msgprio)
int **mq_send** (mqd_t mq, const char *msg, size_t msglen, unsigned int msgprio)
int **mq_close** (mqd_t mq)
int **mq_getattr** (mqd_t mq, struct mq_attr *attrbuf)
int **mq_setattr** (mqd_t mq, const struct mq_attr *new_attrs, struct mq_attr *old_attrs)
int **mq_notify** (mqd_t mq, const struct sigevent *notification)
int **mq_unlink** (mqd_t mq)

“DULCIS IN FUNDO”

(last but far from least)

LXRT

just take ALL the functions (>100) available for UP/SMP/MUP + shared memory + FIFOs and use them inter-intra RTAI modules / LINUX processes

Side effect: better than System V IPC intra LINUX processes (with not so bad performances…)

Within the constrained posed by the LINUX scheduler and non preemptable kernel firm real time possible within LINUX processes (meaning: very good average performances coupled to spikes of unbearable latency; the picture could change once the low latency patch for 2.2.10 becomes standard)

Useful “per se” and for an easier, less risky and faster development phase (the primary reason of its birth)

DRIVERS

- ***SERIAL PORT***
- ***STANDARD PARALLEL PORT***
- ***NE2000 ETHERNET CARD***
- ***AD/DA CARDS*** (Intelligent Instrumentation PCI20428W, Bluechip Technology ADC-44, Advantech PCL-818HG/HD and PCL727, Keithley DAS 1600)

+

Commitment to always make available all what distributed with NMT-RTL (if and when required, provided RTL remains and adds (L)GPL)

IMMEDIATE FUTURE

- an eye on 2.3.xx to be ready for 2.4.xx (at the moment 2.3.xx seems to make RTHAL-RTAI easier)
- a lot of refiniments
- an eye on APIs
- hope for a lot of contributions
- fading hope to see the RTHAL concept native in LINUX
- porting RTHAL-RTAI to other architecture (if someone pays while leaving it all LGPL)

NON DIAPM ACKNOWLEDGMENTS (I)

(at DIAPM they know all my debts for their invaluable help already)

D. Beal
V. Brushkoff
P. Cloutier
P. Daly
D. Danlugli
R. Finazzi
S. Hughes
B. Knox
J. Kuepper
K. Kumsta
S. Papacharalambous
D. Schleef
C.Schroeter
C. Tannnert
P. Wilshire
T. Woolven

(those deserving a shuffling of the above surname sort already know the value of their help)
(my sincere apologies for anyone forgotten)

NON DIAPM ACKNOWLEDGMENTS (II)

M. Barabanov and V. Yodaiken

(… but DIAPM-RTAI has nothing to do with FSMLabs-RTL)